Microsoft® .NET

Market Bulletin

.NET Server Solutions

Group • July 2001

The Middleware Company Shows Their Bias

In June 2001, Ed Roman & Chad Vawter of The Middleware Company published a 25-page analysis entitled "J2EE vs. Microsoft.NET: A comparison of building XML-based web services".[1] The paper offers a generally solid top-level description of what the industry means by "Web Services", and the technologies required to make Web Services real. Their position is that Web Services will define the next generation of distributed computing. On this much we agree. However, the analysis includes a number of misstatements and inaccuracies related to both Microsoft's products and technologies, and Sun's technologies. This bulletin offers comments and corrections to that analysis.

J2EE has NOT been extended to support Web Services

First, early on, the paper states that J2EE has been extended to support Web Services. This is absolutely untrue. The current J2EE and Java specifications say nothing about Web Services. While Sun, the owner of the J2EE specifications, is planning to extend J2EE with several new JSRs[2] relating to Web Services, there are currently no Java specifications, or even proposed specifications in public draft form, directly supporting Web Services. To be accurate, some *vendors* are extending their products to support Web Services, but this is product innovation, and has nothing to do with the J2EE specifications. The paper seems to suggest the Java APIs for XML (JAXP) are Java APIs relating to Web Services. This is a big stretch. JAXP is an XML parsing API. There is much more to developing, hosting, and consuming Web Services than simply parsing XML documents. The key Web Services technologies mentioned in the paper, such as SOAP, WSDL, UDDI, and so on, are currently absent from J2EE.

Extra Features are Vendor-specific

Aside from confusing an XML parser with a Web Services engine, the paper gives a concise review of J2EE technologies. However, one summary comment seems unfounded. The paper states that features such as load-balancing, failover, and caching do not affect portability of application code across different vendor implementations. This is true in some cases, not true in others (see, for instance, the caching APIs in iPlanet's Application Server). But a more importantly, this point is irrelevant. If real portability of applications is the goal, then the analysis cannot be confined to examining only the portability of "application code". What about the application metadata? What about installation and operational procedures? What about deployment practices? All of these are vendor-specific (not covered in J2EE); the combined result is that real applications are not portable from one vendor's product to another, even if some portion of the application code itself is portable. IBM, one of the vendors who claims support for J2EE, which supposedly confers "portability" on apps, confirms this by publishing a 274-page

[1] http://www.theserverside.com/resources/article.jsp?l=J2EE-vs-DOTNET

[2] JSR = Java Specification Request. This is the mechanism by which new technology APIs are proposed and adopted into Sun's Java technology.

document describing how to migrate applications from a competitor's product to their product.[3]

.NET does not REPLACE existing technology

Moving on to the paper's description of the Microsoft technologies, there are many points that warrant clarification.

The paper states that Microsoft's .NET Framework *replaces* other Microsoft technologies such as COM+, MSMQ, and SQL Server. This is not accurate. In fact, none of these three technologies are replaced by the .NET Framework. The .NET Framework does provide new ways for programmers to access these resources. For example, the .NET Framework makes it easier for applications to take advantage of the COM+ enterprise services. The technologies themselves are not replaced, so COM+, MSMQ, and SQL Server all will continue to be shipped going forward.

On Thick vs Thin Clients

The paper also states that "thick clients" in the Microsoft framework connect to ASP.NET pages. This is surely an editing error – it seems more accurate to say that "thin clients" use ASP.NET, while "thick clients" or "rich clients" can use Windows Forms. That being said, there is nothing that would prevent an app that uses Windows Forms to also connect to resources through remote ASP.NET pages.

MSIL is Never Interpreted

The paper describes Microsoft's Intermediate Language (MSIL) and states that it is interpreted. In fact, in the Microsoft .NET Framework, MSIL is *never* interpreted, nor was it designed to be interpreted. In contrast to some other p-code based systems, IL is always compiled before being executed. Sometimes it is JIT compiled, sometimes it is pre-compiled. But it is never interpreted, at least not by Microsoft's platform offering. There are 3rd party MSIL interpreters, for example DevelopMentor has made one available, but none are currently available directly from Microsoft.

.NET Does not Lack Key Features

The paper states that J2EE has features such as persistence, programmatic transactions, state management, and custom tags that may offer time-to-market benefits. The paper also states that Microsoft.NET lacks these features. This is not true. In fact:

- It is possible to define and construct custom ASP.NET tags using Microsoft's .NET Framework. This is similar to custom tags in JSP.
- The data access technology in .NET, called ADO.NET, provides an object model for automatic persistence, with Create, Read, Update and Delete methods that are automatically generated, but can be overridden by the programmer if desired.
- .NET provides state management, including http session state persistence in ASP.NET, and stateful components.
- .NET allows programmatic transaction management. Managed applications running on .NET can take advantage of all COM+ enterprise services, including transactions.

In a later paragraph, the paper indicates that developers need to write unmanaged code in order to take advantage of transactions or stateful components. This also is inaccurate. Developers can build transactional components in .NET using only managed

[3] http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/sg245956.html

code (for example, code written in C# and running in the CLR). There is no need to write unmanaged code.

.NET Excels in Integration

In comparing the legacy integration capabilities of J2EE and .NET, the paper's description of .NET's capabilities is limited to only the Host Integration Server. While Microsoft's HIS is a valuable product, it is by no means the only way to integrate with legacy assets from within the Microsoft platform. On the contrary, there are a wide variety of COM connectors already built, proven, and running for external systems, including such systems as:

- IBM's MQSeries
- SAP R/3
- Siebel applications
- AS/400 filesystems
- Oracle applications
- J.D. Edwards
- PeopleSoft
- And many, many others.

Since the .NET Framework is fully interoperable with existing COM components, all of these integration capabilities and features carry forward with .NET. The authors of the paper have failed to consider these key capabilities, and have incorrectly concluded that Microsoft's platform somehow does not offer excellent integration capabilities.

Aside from the excellent COM-based integration options, .NET offers BizTalk Server, which has received great glowing reviews from analysts and customers for its power to integrate disparate systems. There is nothing currently in J2EE to support orchestration of business processes, although, again, Sun says it has plans to extend J2EE in this direction.

JCA is Not Real

One additional point on integration: the paper describes the Java Connector Architecture as a strength for J2EE-based systems. However, JCA is a still-evolving paper specification. None of the enterprise application systems cited in the paper actually offer production-ready JCA-based connectors today. While JCA may hold promise for the Java community, it is currently more vision than reality. On the other hand, JCA is based on a tightly-coupled model, one that requires Java on the requestor side. It is in direct contrast to the loosely-coupled model of Web Services, which is, ironically, the topic of the paper in question. Sun's investment in tightly-coupled connectors seems to be counter to reason, and the broader trend in the industry toward loose coupling and Web Services. The paper fails to mention this.

Language Support in .NET

In the treatment of language support, the paper states that .NET will not support Java. Microsoft has gone on record as stating that the Java language will be a supported language on the .NET Framework (see Bill Gates' statements at Comdex in November 2000). While Microsoft has not yet released this technology, it is coming. To be very clear, Microsoft is not supporting the Java *platform* – including all the APIs specified and owned by Sun. Instead .NET will support the Java *language* on the .NET platform. This may or may not qualify as "Java support", depending on the point of view of the reader.

No questions about Multi-platform for .NET

Regarding multi-platform support for .NET, contrary to statements in the paper, Microsoft has made no claims about the availability of .NET on non-Microsoft platforms. We want

customers and the industry to understand that .NET is fundamentally a Microsoft platform.

Visual Studio is the Best Tool

In a rare moment of lucidity and honesty, the authors admit that the tools in the Java market for building Web Services are low quality, and not integrated with the rest of the toolset. The authors state "Microsoft has the clear win when it comes to tools." We agree!

.NET is Fully standards-based

The paper compares Web Services infrastructure, and questions Microsoft's commitment to open web services standards. Microsoft is driving and helping to define the most important standards around Web Services, including XML, SOAP, WSDL, and UDDI. We are working with industry organizations such as W3C and with industry partners such as IBM to define and document these standards. Furthermore, we work with these industry partners in an open forum to test and demonstrate cross-vendor Web Services interoperability. We continue to monitor other specification efforts, and will continue to participate in all open efforts.

.NET Leads in Performance

In the discussion of performance and scalability, the paper restates some inaccuracies previously mentioned, around state management, and caching. .NET does in fact have state management and caching capabilities, and these capabilities are accessible to application developers. The analysis delivered by the paper here relies heavily on theory, and very little on practical observations. In fact, J2EE has never been shown to deliver leading performance or scalability, when compared with alternative technologies in public benchmarks. Public benchmarks such as the TPC-C and the TPC-W, and even the Web Application Server test done by PC Magazine in May 2001, show that Microsoft's platform leads on performance and price/performance. So the comments Vawter and Roman make on performance and scalability look like vigorous hand waving, in the face of contrary evidence.

On Editorial Independence

Two final comments: in the preface of the article, the authors promise to remain unbiased. This is an admirable and ambitious position to take, especially considering:

- The cover page states that the paper was prepared for Sun Microsystems, the owner of the J2EE specification, and obviously a party with vested interest in the conclusions the paper draws.
- The authors' employer, The Middleware Company, has partnerships with Java vendors, a J2EE consulting and training practice, and a strong vested interest in the success of J2EE as a platform.

Could it be that, somehow, inadvertently, imperceptibly, these factors may have influenced the review? Could it be that the business relationships of The Middleware Company have colored the supposedly "unbiased" analysis and conclusions in this paper ? For example, the conclusion that .NET is a monopolistic initiative arises from the ether, without any supporting argument in the paper. The questioning of Microsoft's commitment to standards also seems illogical. The statements that J2EE is a leader in performance are contrary to all available public evidence. The conclusion that "J2EE will begin to dominate more and more as time goes on" is offered without any supporting argument, and in direct contrast to the views held by many independent analysts. For

example, analysts at the Gartner Group in May 2001 described Microsoft as the leading vendor in Web Services.[4] And Gartner believes that J2EE will not dominate. [5]

The Paper vs. Product Problem

Second, the comparison struggles with the familiar "paper vs. product" problem – it compares paper specifications from Sun with actual products from Microsoft. While this is an interesting exercise, at some point the approach fails to deliver a real view of practical options, because of the significant differences in vendor product offerings, even when those products are certified as J2EE compliant. Remember, *J2EE says nothing about Web Services*, so for Web Services, the capabilities of a product are quite vendor-specific. The problem also arises when considering such issues as development tools, pricing, scalability, and so on. These are all topics mentioned in the paper, but without vendor-specific analysis, conclusions aren't worth much. A clear understanding of options and capabilities can only be accomplished by comparing real products to real products.

Summary

We at Microsoft agree that the future of computing is Web Services. We've worked very hard on making .NET the best platform on which to build, deploy, and run Web Services applications. Do a fair comparison, and we think you'll agree.

[4] See Gartner document # IGG-05232001-01
[5] http://www.gartner.com/webletter/microsoft/article10/article10.html